同济大学交通运输工程学院
COLLEGE OF TRANSPORTATION ENGINEERING
TONGJI UNIVERSITY

交通数据分析

第十二讲 经典深度学习结构

沈煜 博士 副教授
嘉定校区交通运输工程学院311室
yshen@tongji.edu.cn
2022年05月13日

计划进度

周	日期	主讲	内容	模块
1	2022.02.25	沈煜	概述	
2	2022.03.04	沈煜	在线数据采集方法	爬虫
3	2022.03.11	沈煜	线性回归模型	回归分析
5	2022.03.18	沈煜	广义线性回归	
4	2022.03.25	沈煜	广义线性回归（作业1）	
6	2022.04.01	沈煜	空间数据描述性分析	
7	2022.04.08	沈煜	空间自回归方法（作业2）	
8	2022.04.15	沈煜	关联：Apriori	机器学习
9	2022.04.22	沈煜	决策树、支持向量机（作业3）	
10	2022.04.29	沈煜	浅层神经网络	
11	2022.05.06	沈煜	卷积神经网络（期末大作业）	
12	2022.05.13	沈煜	经典网络结构	
13	2022.05.20	沈煜	聚类：K-Means、DBSCAN	
14	2022.05.27	沈煜	贝叶斯方法、卡尔曼滤波	
15	2022.06.03	-	端午节放假	
16	2022.06.10	沈煜	期末汇报（1）	
17	2022.06.17	沈煜	期末汇报（2）	

主要内容

- 经典CNN结构
- RNN简介

AlexNet

- 以第一作者Alex Krizhevsky的名字命名
  - Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks
  - 这篇文章向计算机视觉领域证明了深度学习的重要意义

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

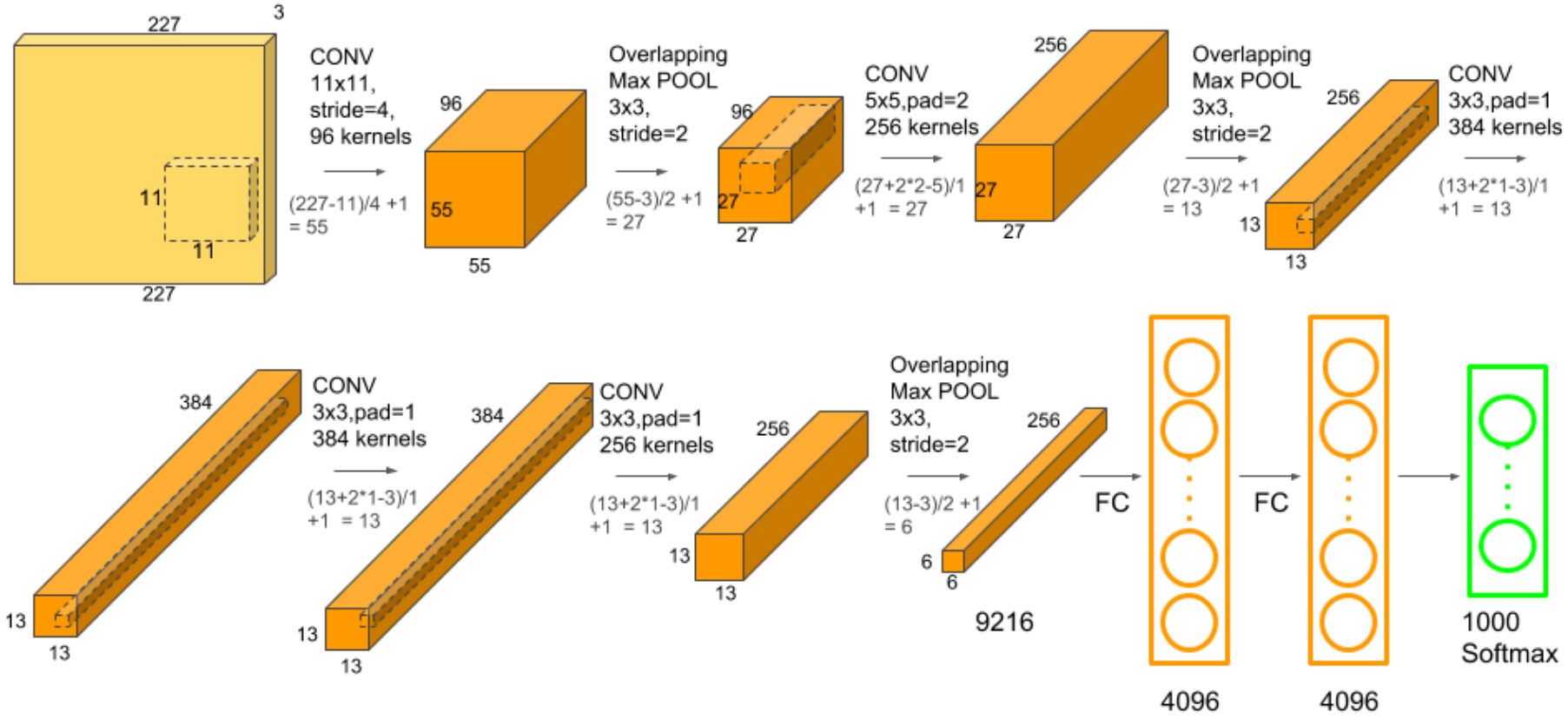
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

AlexNet

Conv => Max-pool => Conv => Max-pool => Conv => Conv =>
Conv => Max-pool => Flatten => FC => FC => Softmax

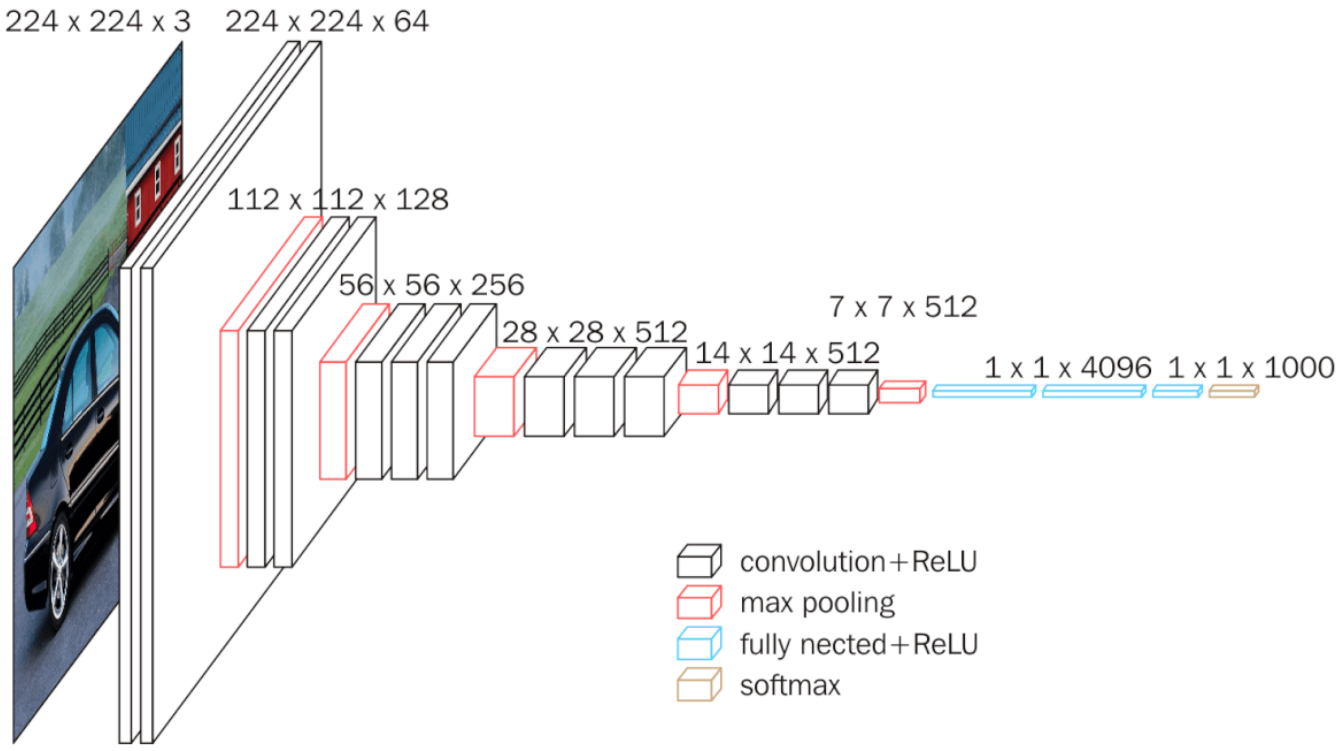


AlexNet

- 与LeNet-5结构相似，但是网络结构大的多
- 有6000万个参数
  - LeNet-5只有6万个参数
- 使用ReLU作为激活函数
- 需要使用多颗GPU进行运算

VGG-16

- 在AlexNet基础上的升级
  - Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition



VGG-16

- 减少了超参数的数量
  - 所有卷积核函数3×3，步长为1
  - 最大池化2×2，步长为2
- 1亿3千8百万个参数
  - 全连接层参数多
- 消耗内存巨大
  - 每向前传播一次需要96M的内存
- 卷积核（过滤器）数量递增
  - 64->128->256->512->512
- 只通过池化降低数据维度
- 为CNN结构的建立提供了一些经验和规则

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

GoogLeNet

- ILSVRC 2014：Inception V1
- 22层，只有500万的参数量
  - 去除最后的全连接层，取代以全局平均池化层（即将图片尺寸变为1*1）
  - 1*1的卷积：跨通道组织信息，提高网络的表达能力，同时可以对输出通道升维和降维
  - 全连接层几乎占据了AlexNet或VGGNet中90%的参数量，并可能引起过拟合

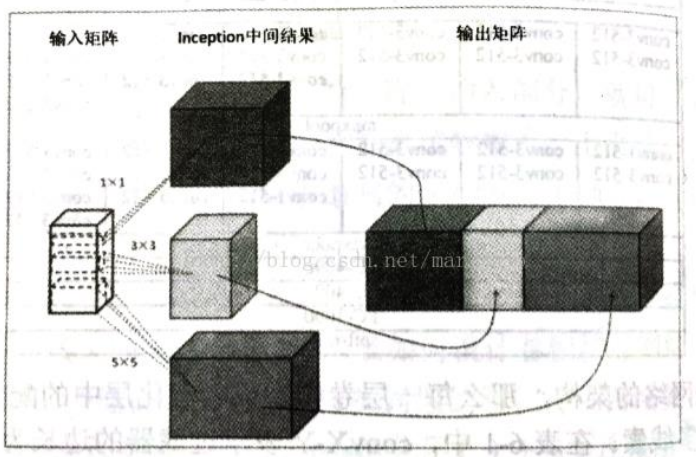
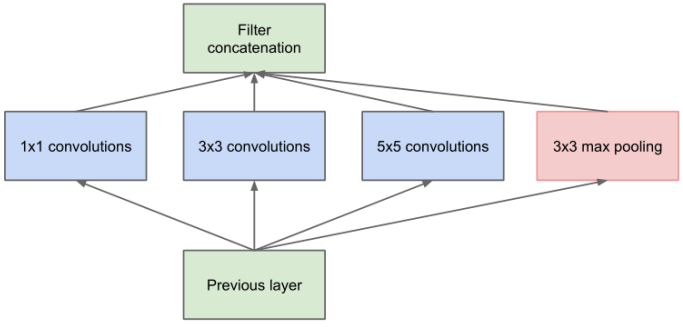
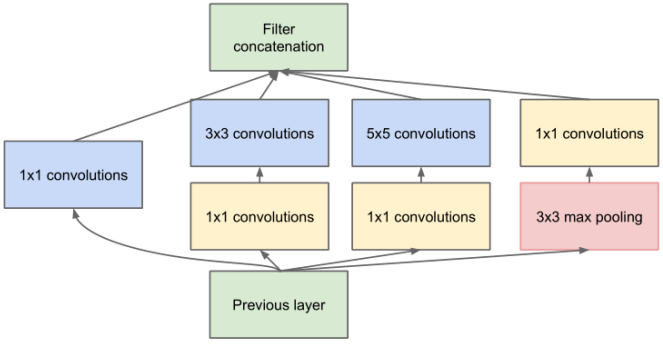


图 6-16 Inception 模块示意图。

- Inception Module

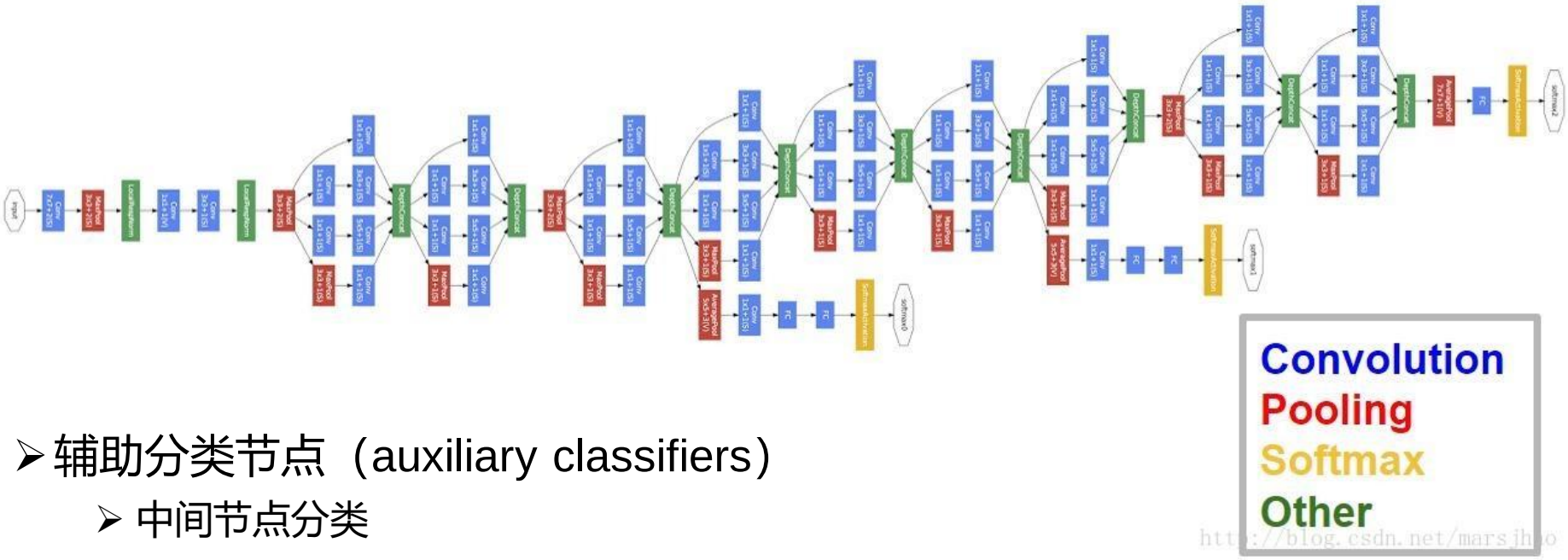


(a) Inception module, naïve version



(b) Inception module with dimension reductions

GoogLeNet

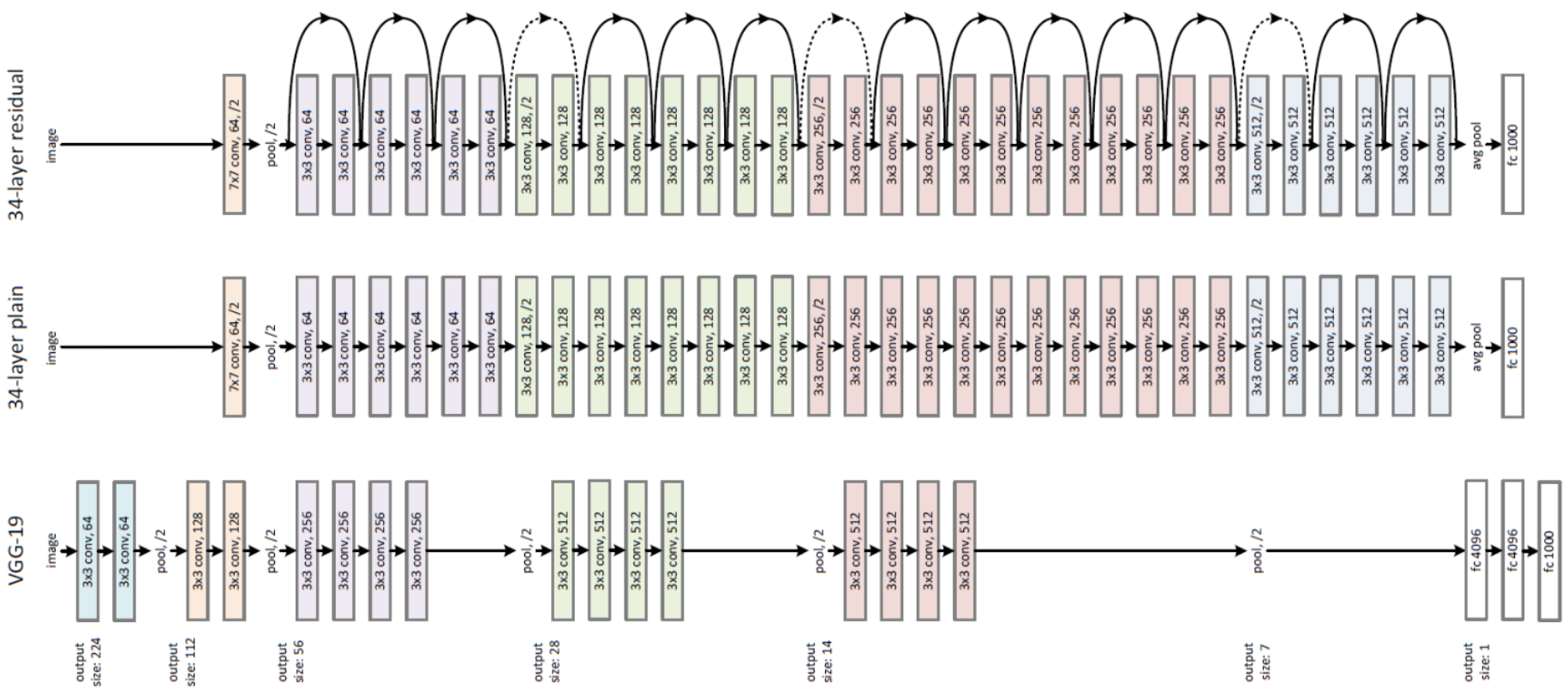


- 辅助分类节点（auxiliary classifiers）
  - 中间节点分类
- 模型融合
  - 将中间某一层的输出用作分类，并按一个较小的权重（如0.3）加到最终分类结果中
- 实际的GoogLeNet在最后加了一个全连接层
  - 为了方便以后finetune

ResNet

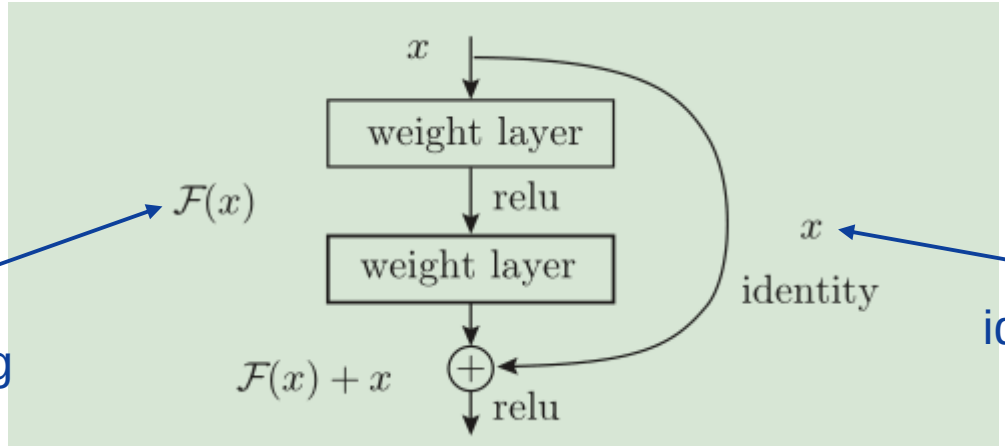
- 2015年前的普遍观点
  - CNN结构的层数越多，表述能力越强（对特征的描述能力也就越强），效果也就越好
- 何恺明
  - Deep Residual Learning for Image Recognition，2015
- 退化现象（degradation problem）
  - 由于层数的增加，网络很深的常规网络（plain network）有时候的效果反而不如浅层网络（shallower network）
  - 不是所有的神经网络结构都是容易收敛和优化的
- 梯度消失/爆炸（vanishing/exploding gradient）

ResNet


VGG-19
image
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 224
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 112
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 56
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 7
fc 4096
fc 4096
fc 1000
output size: 1
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000
34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

ResNet

- 网络设计：
  - 快捷结构（shortcut）
  - 全部（或者大部分采用）采用3×3的卷积核
  - Spatial size /2 -> filter×2 （保证时间复杂度一致）
  - 没有Dropout层
- ResNet能够在没有任何困难的情况下得到训练，并且实现更深的网络结构使其达到更低的训练误差



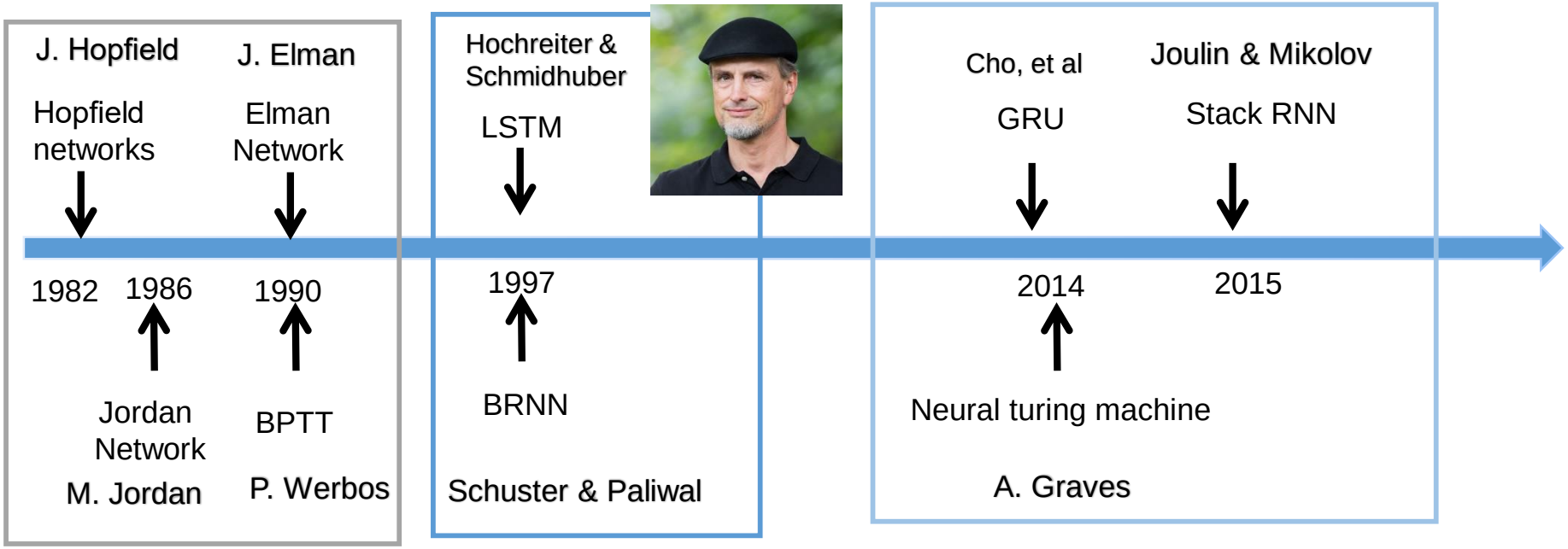
循环(递归)神经网络

RNN

RNN的意义

- 传统神经网络，包括CNN，输入和输出都是相互独立的
  - 如：不同的图像之间是分隔独立
  - 但是，有些任务中，后续的输出和之前（或之后）的内容是相关的
  - 自然语言处理的任务中，输入和输出之间不独立
    - “The author of A Song of Ice and Fire is _____”
- RNN引入“记忆”的概念
  - “循环recurrent”来源于每个元素都执行相同的任务
  - 输出依赖于“输入”和“记忆”
    - 做预测不止依赖于输入，还依赖于之前的一部分信息，会把它存在“记忆”中

RNN发展历史



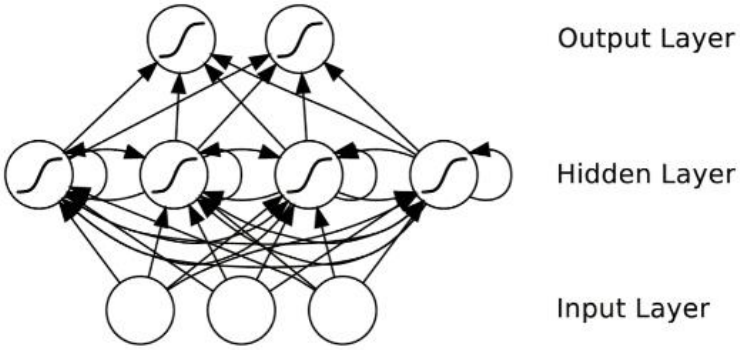
早期（80、90年代）
主要思想：重新使用参数和计算

中期（90-2010）
除LSTM以外，RNN基本从主流研究中消失了。

当前（2010 - ）应用广泛：
自然语言应用
视频建模，手写识别，用户意图预测

开源工具包：
Theano
Torch
PyBrain
TensorFlow
,,,

递归神经网络



- 递归神经网络（RNN），是两种人工神经网络的总称
  - 一种是时间递归神经网络（recurrent neural network）
  - 一种是结构递归神经网络（recursive neural network）

- 吸收了HMM模型的有限序列关联的思想。
- 神经网络的隐藏层结构能够更好的表达有限的观察值背后的复杂分布。

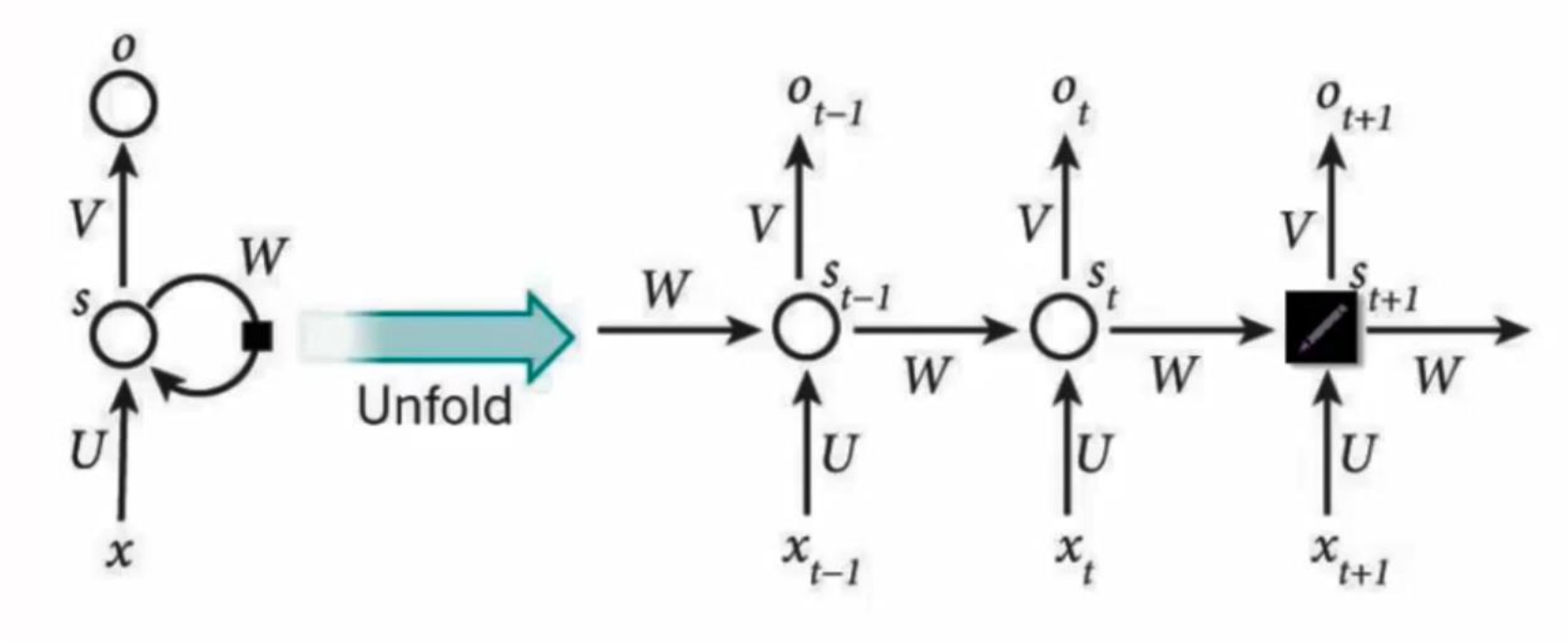
时间递归神经网络

- RNN是一类扩展的人工神经网络，它是为了对序列数据进行建模而产生的。
  - 针对对象：序列数据。例如文本，是字母和词汇的序列；语音，是音节的序列；视频，是图像的序列；气象观测数据，股票交易数据等等，也都是序列数据。
  - 核心思想：样本间存在顺序关系，每个样本和它之前的样本存在关联。通过神经网络在时序上的展开，我们能够找到样本之间的序列相关性。

$$h_t = \mathcal{H}\left(W_{ih}x_t + W_{hh}h_{t-1} + b_h\right)$$

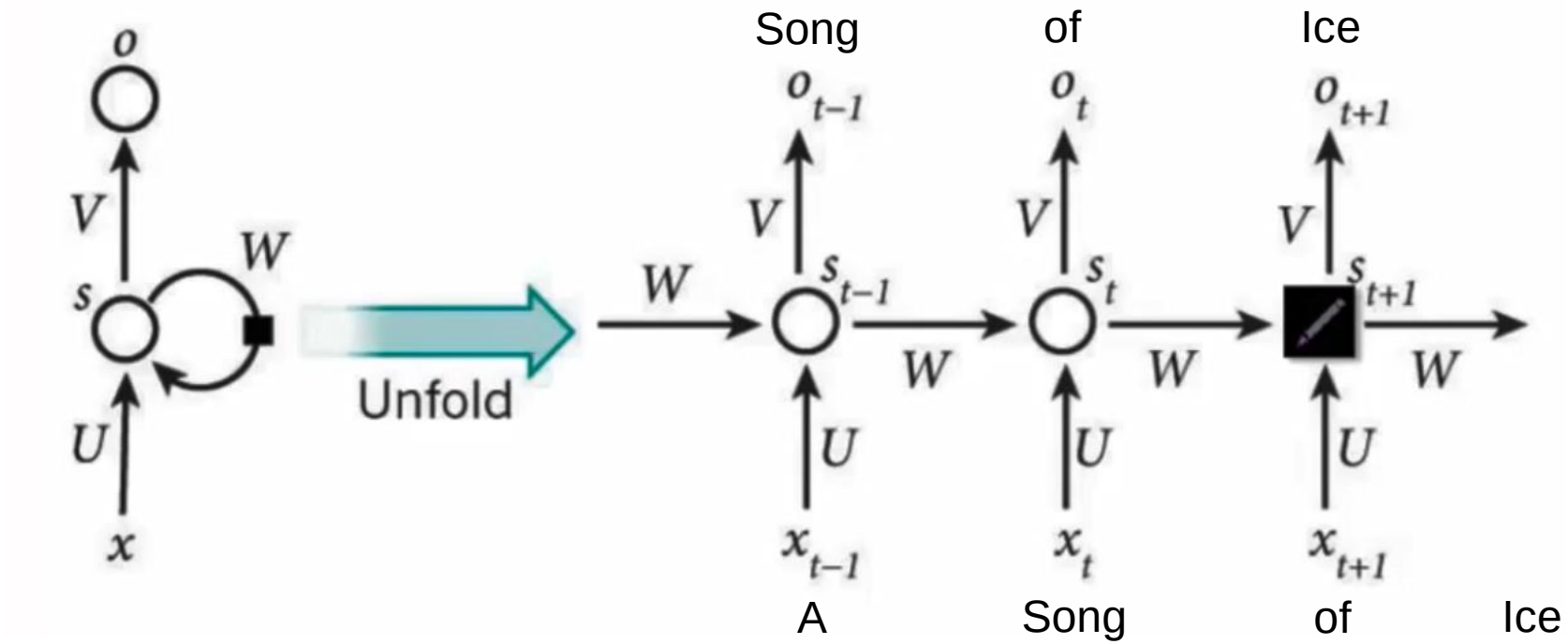
RNN模型基本结构

- 序列按时间展开



RNN模型基本结构

- X_t是时间t处的“输入”
- S_t是时间t出的“记忆”：$S_t = f(UX_t + WS_{t-1})$，f是激活函数，如tanh
- O_t是时间t出的“输出”，比如预测下个词的话，可能是softmax，输出属于每个候选词的概率：$O_t = softmax(VS_t)$

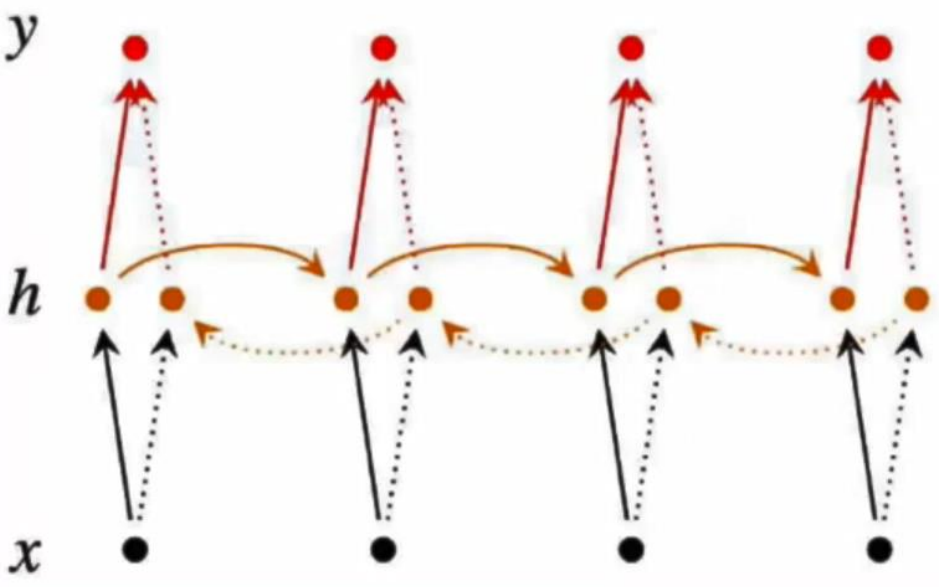


RNN模型基本结构

- 可以把隐状态S_t视作“记忆体”，捕捉了之前时间点上的信息
- 输出O_t由当前时间及之前所有的“记忆”共同计算得到
- 但是，实际应用中，S_t并不能捕捉和保留之前所有的信息（记忆力有限）
- 与CNN不同，RNN中，整个神经网络都共享一组参数（U, V, W），极大减小了需要训练和预估的参数量
- 在有些任务下，O_t是不存在的
  - 比如文本情感分析，只需要最后的输出结果

双向RNN

- 有些情况下，当前的输出不只依赖于之前的序列元素，还可能依赖于之后的序列元素
  - 比如“The ____ of A Song of Ice and Fire is George RR Martin”
- 直观理解：双向RNN叠加



前向后的参数

$$\overrightarrow{h_t} = f\left(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h_{t-1}} + \vec{b}\right)$$

后向前的参数

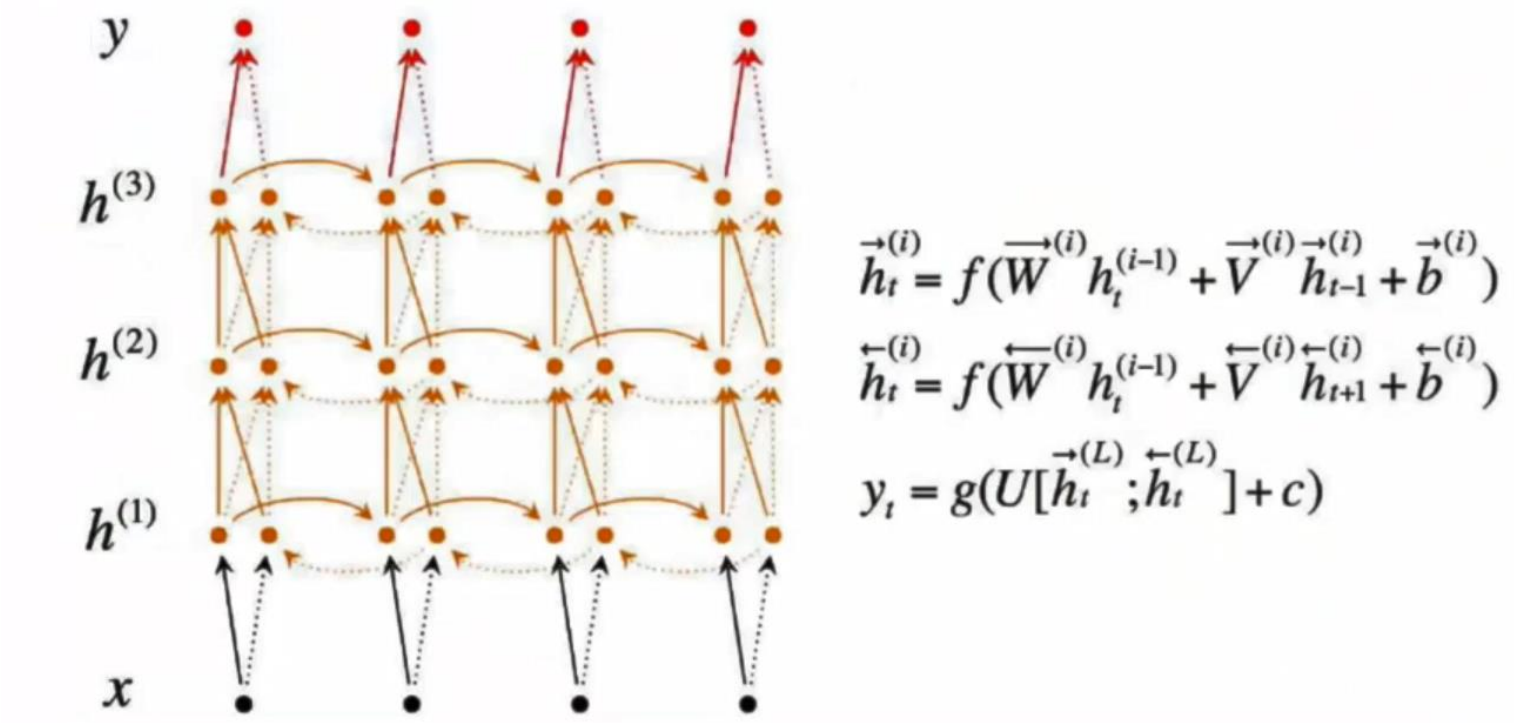
$$\overleftarrow{h_t} = f\left(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h_{t+1}} + \overleftarrow{b}\right)$$

两个记忆做拼接

$$y_t = g\left(U\left[\overrightarrow{h_t}; \overleftarrow{h_t}\right] + c\right)$$

深层双向RNN

- 区别于双向RNN，每一步/每个时间点都是多层结构

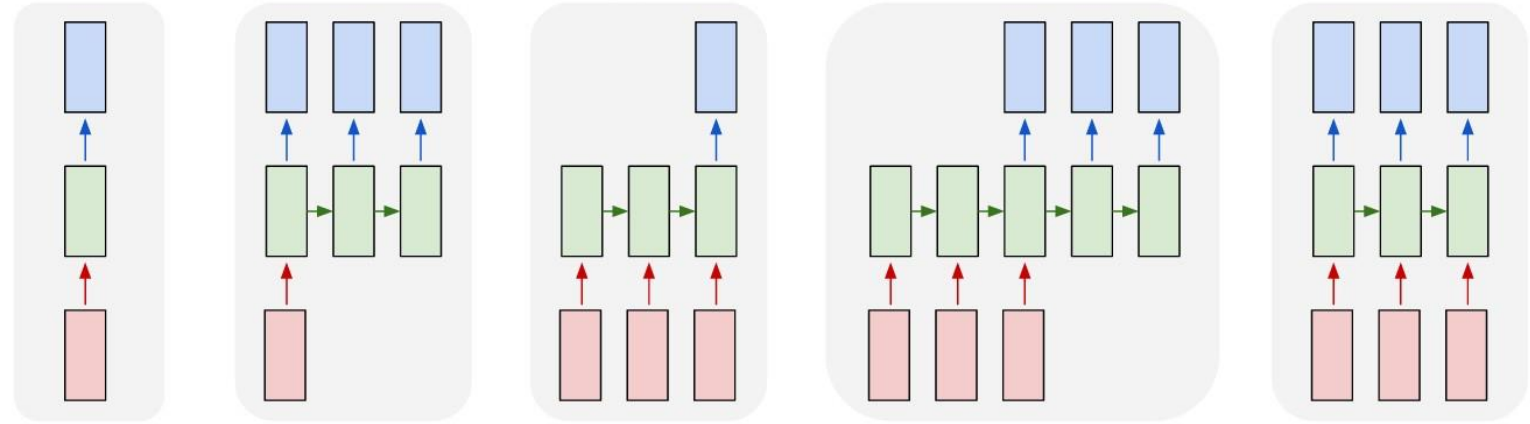


$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

RNN基本类型



如：文本情感分析
系列文本->情感

如：视频按帧分类

如：图像标注
图片->系列文本

如：机器翻译
系列文本->系列文本

图像标注

"man in black shirt is playing guitar."

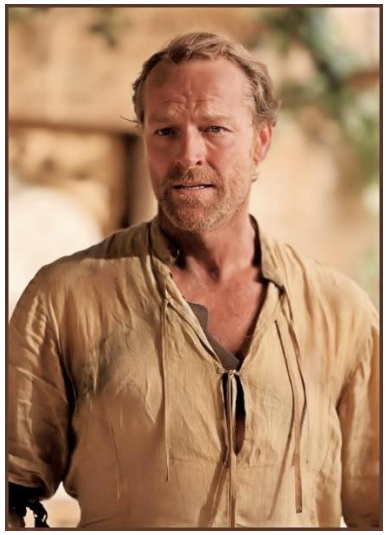
"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

语言生成

GOT Book 6 Generator

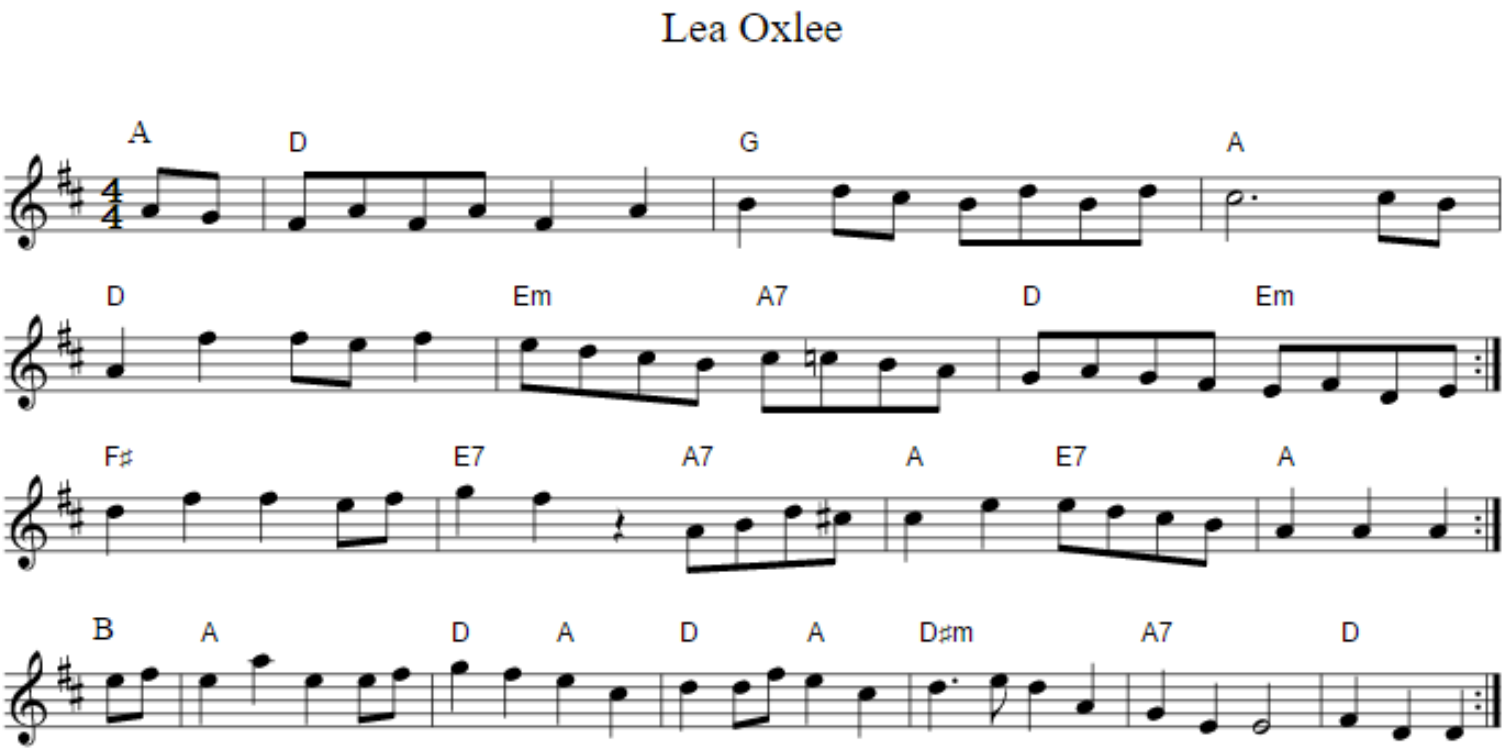
Are you tired of waiting for the next GOT book to come out? I know that I am, which is why I decided to train a RNN on the first five GOT books and use predictions from the network to create the sixth book in the series. The first five chapters of the generated sixth book are now available and are packed with as many twists and turns as the books we've all come to know and love. Here's the sparknotes summary:



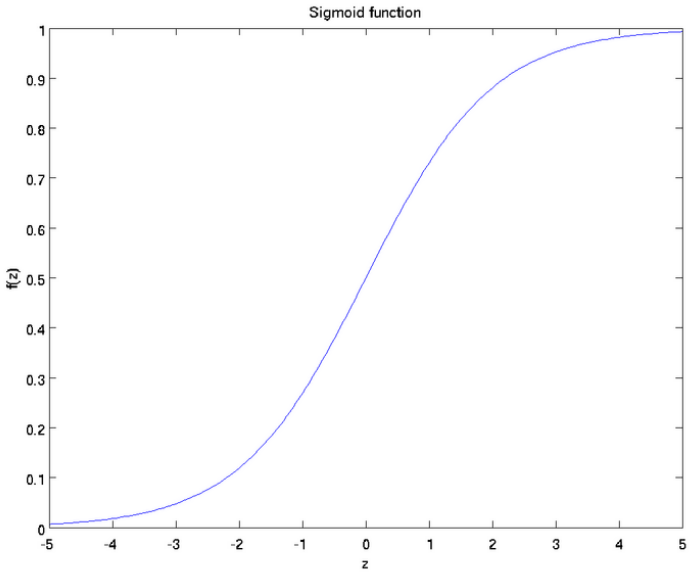
A hundred yards east, Ser Jorah lingered to where the banners wending their descent down a long ways of rain. The marsh was ladling out beef-and-barley stew, cold as shy of three colors, chunks of butter.”

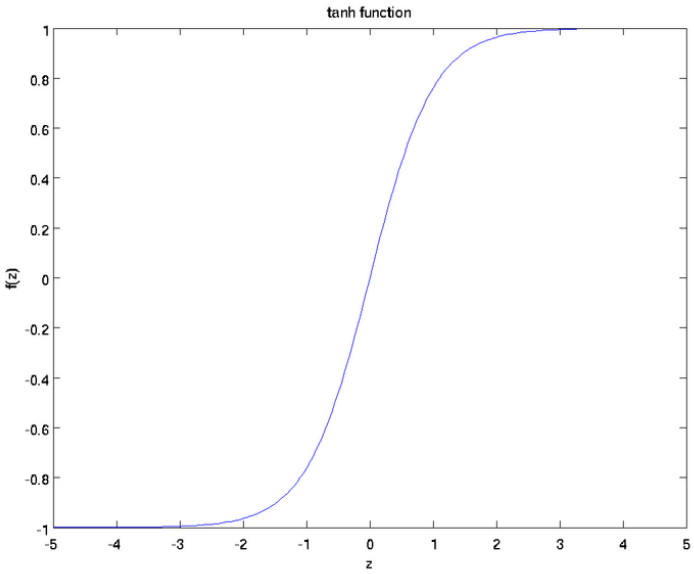
“往东一百米，乔拉爵士在那踌躇徘徊。低垂的旌旗蜿蜒在漫长的雨中。沼泽中冒出牛肉大麦炖汤，冷若害羞的三种颜色和成块的黄油”

音乐作曲



RNN常用的激活函数





$$f(z) = \frac{1}{1 + \exp(-z)}$$

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

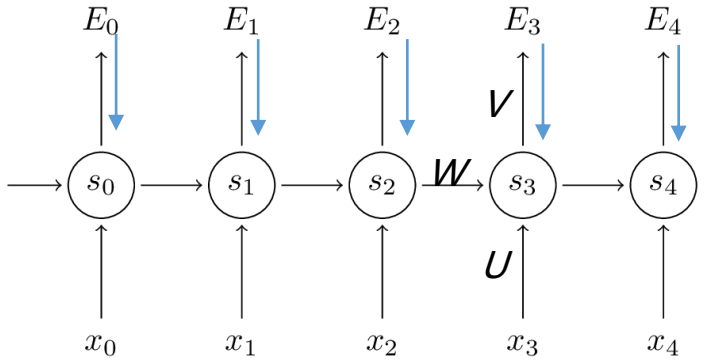
Softmax

- Softmax函数是sigmoid函数的一个变种，通常我们将其用在多分类任务的输出层，将输入转化成标签的概率。
- 本质就是将一个K维的任意实数向量压缩（映射）成另一个K维的实数向量，其中向量中的每个元素取值都介于（0，1）之间。

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)}=1|x^{(i)};\theta) \\ p(y^{(i)}=2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)}=k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

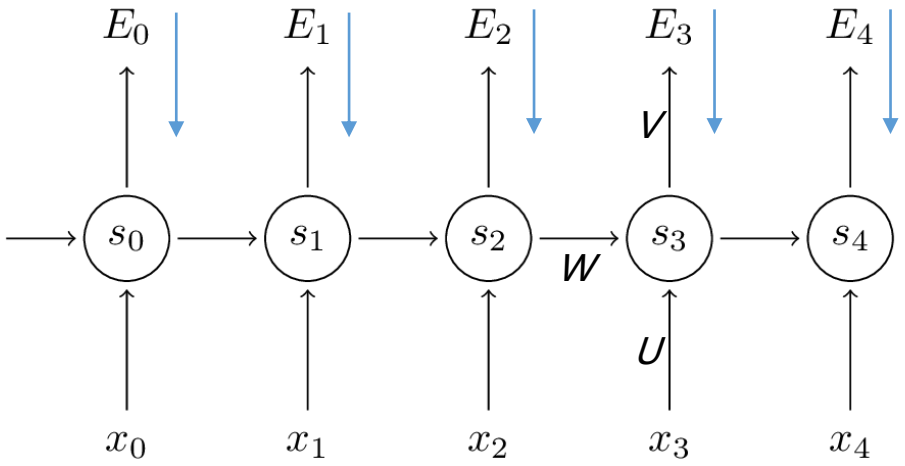
随时间反向传播算法BPTT

- **BP回顾：**定义损失函数E来表示输出$\hat{y}$和真实标签y的误差，通过链式法则自顶向下求得E对网络权重的偏导
- 沿梯度的反方向更新权重的值，直到E收敛。
- BPTT的本质其实和BP很像，就是加上了时序演化
- 定义权重U，V，W
  - $s_t = \tanh(Ux_t + Ws_{t-1})$
  - $\hat{y}_t = \mathrm{softmax}(Vs_t)$
- 定义损失函数:
  - $E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$
  - $E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$
- 我们将整个序列作为一次训练，所以需要对每个时刻的误差进行求和



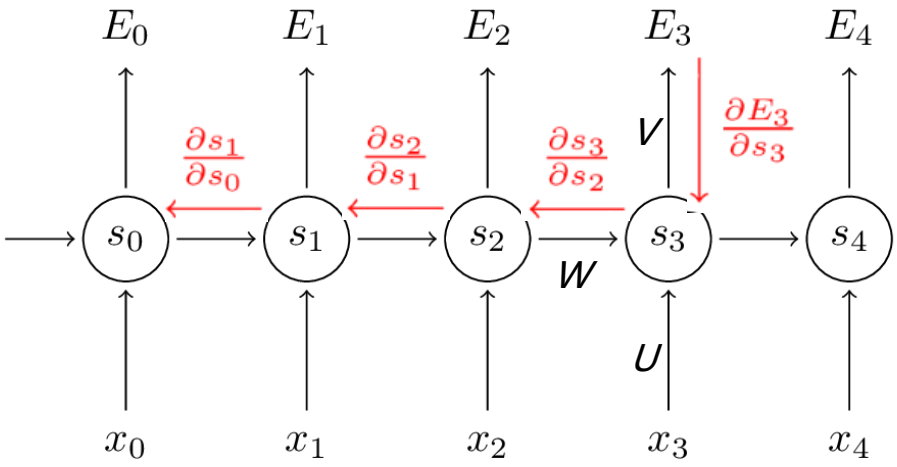
随时间反向传播算法BPTT

- 目前的任务是求E对于U, V, W的梯度。
- 定义E对于W的梯度
  - $\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$
  - U, V同理
- 求E对于V的梯度
- 例：求E_3对于V的梯度
  - $\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V}$
  - 其中$z_3 = Vs_3$
- 求和可得$\frac{\partial E}{\partial V}$



随时间反向传播算法BPTT

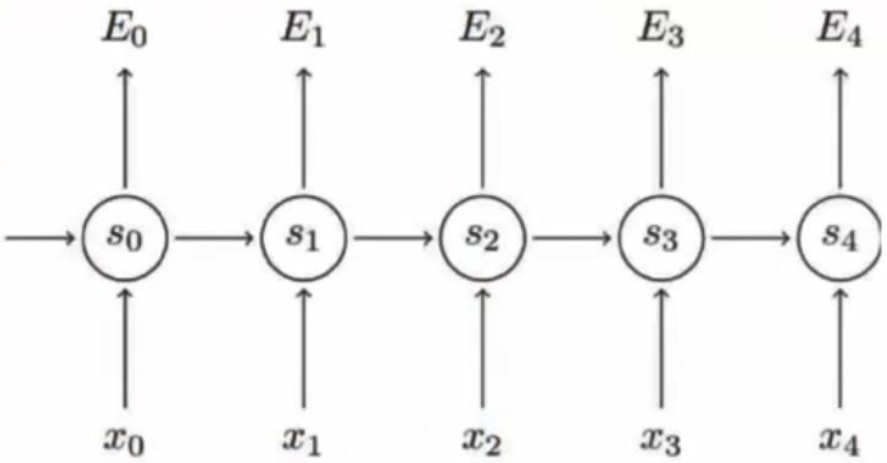
- 求E对于W的梯度（注意，现在情况开始变得复杂起来）
- 先求E_3对于W的梯度
- $\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$
- 当我们求s_3对于W 的偏导时
- 注意到：
- $s_3 = \tanh(Ux_3 + Ws_2)$
  - 其中：s_3依赖于s_2
  - 而s_2又依赖于s_1和W
  - 依赖关系一直传递到$t = 0$的时刻



- 因此，当我们计算对于W的偏导数时，不能把s_2看作是常数项
- $\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3}\frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$；求和可得$\frac{\partial E}{\partial W}$

随时间反向传播算法BPTT

- 计算误差关于参数U、V和W的梯度，然后使用梯度下降法学习出好的参数。



交叉熵损失
cross entropy loss

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

每一个时间点都有一个输出
每个输出都可以去计算loss

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$$

梯度下降，使loss值最小
要求我们去求loss function对于W的偏导

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

求偏导的过程在这一层完成不了，必须用BPTT，不能用BP

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

但是 $s_3 = \tanh(Ux_t + Ws_2)$ 依赖于s_2

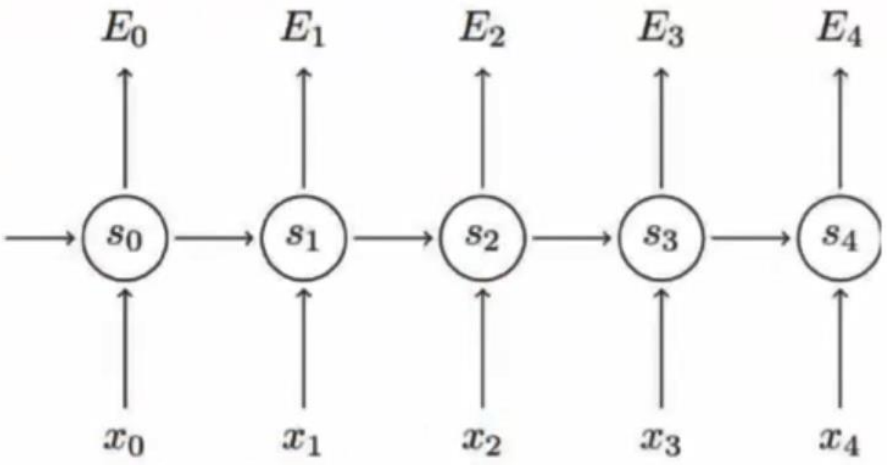
BPTT
沿着时间轴往前追溯

链式法则

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

随时间反向传播算法BPTT

- 输出是Softmax的分类器，预测4W个词中的哪一个



t时刻上正确的词

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

t时刻上预测出来的词

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$$

完成整个句子预测：沿着时间轴把每个位置的 loss加在一起

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

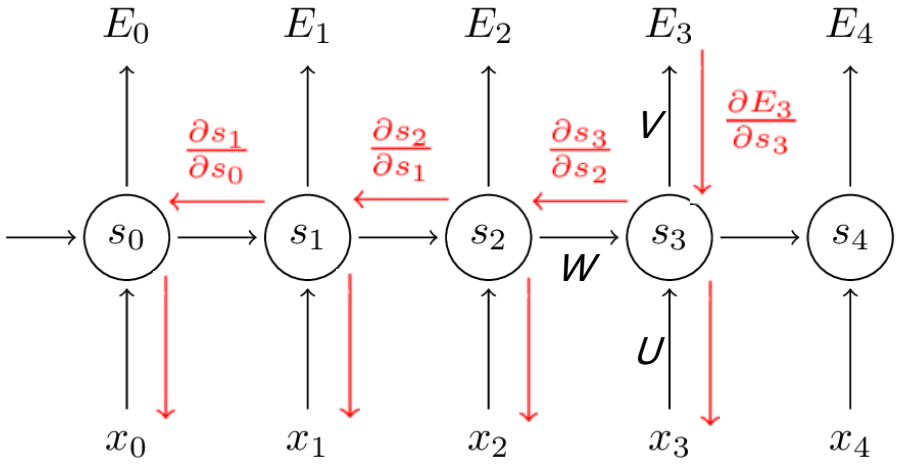
但是 $s_3 = \tanh(Ux_t + Ws_2)$ 依赖于s_2

链式法则

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

随时间反向传播算法BPTT

- 求E于U的梯度（情况与W类似）
- 先求E_3对于U的梯度
- $\frac{\partial E_3}{\partial U} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial U}$
- 当我们求s_3对于U的偏导时
- 注意到：
- $s_3 = \tanh(Ux_3 + Ws_2)$
  - 同样，s_3依赖于s_2
  - 而s_2又依赖于s_1和U



- 类似求W，当我们计算对于U的偏导数时，也不能把s_2看作是常数项
- $\frac{\partial E_3}{\partial U} = \sum_{k=0}^{3}\frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial U}$；求和可得$\frac{\partial E}{\partial U}$

随时间反向传播算法BPTT

for t **from** T **downto** 1 **do**

$do_t \leftarrow g'(o_t) \cdot dL(z_t; y_t)/dz_t$

$db_o \leftarrow db_o + do_t$

$dW_{oh} \leftarrow dW_{oh} + do_t h_t^\top$

$dh_t \leftarrow dh_t + W_{oh}^\top do_t$

$dz_t \leftarrow e'(z_t) \cdot dh_t$

$dW_{hv} \leftarrow dW_{hv} + dz_t v_t^\top$

$db_h \leftarrow db_h + dz_t$

$dW_{hh} \leftarrow dW_{hh} + dz_t h_{t-1}^\top$

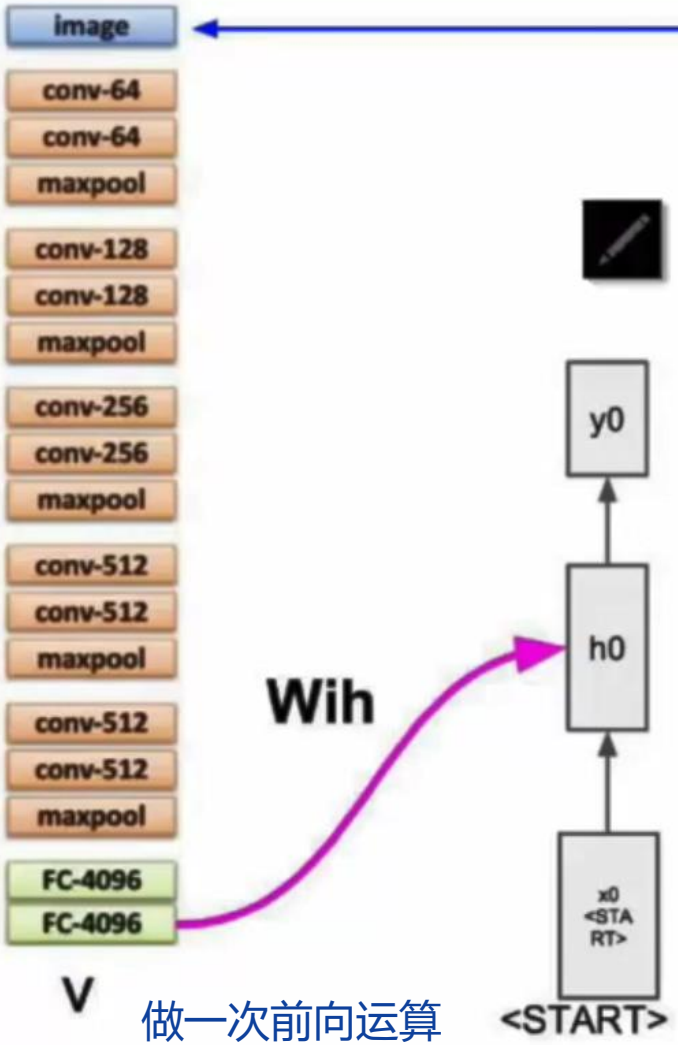
$dh_{t-1} \leftarrow W_{hh}^\top dz_t$

end for

Return $d\theta = [dW_{hv}, dW_{hh}, dW_{oh}, db_h, db_o, dh_0]$.

- 参数意义:
  - W_{hv}:输入层到隐含层的权重参数，
  - W_{hh}:隐含层到隐含层的权重参数，
  - W_{oh}: 隐含层到输出层的权重参数，
  - b_h:隐含层的偏移量,bo输出层的偏移量，
  - h_0:起始状态的隐含层的输出，一般初始为0。

RNN与图像描述



test image

before:

$h = \tanh(W_{xh} * x + W_{hh} * h)$

now:

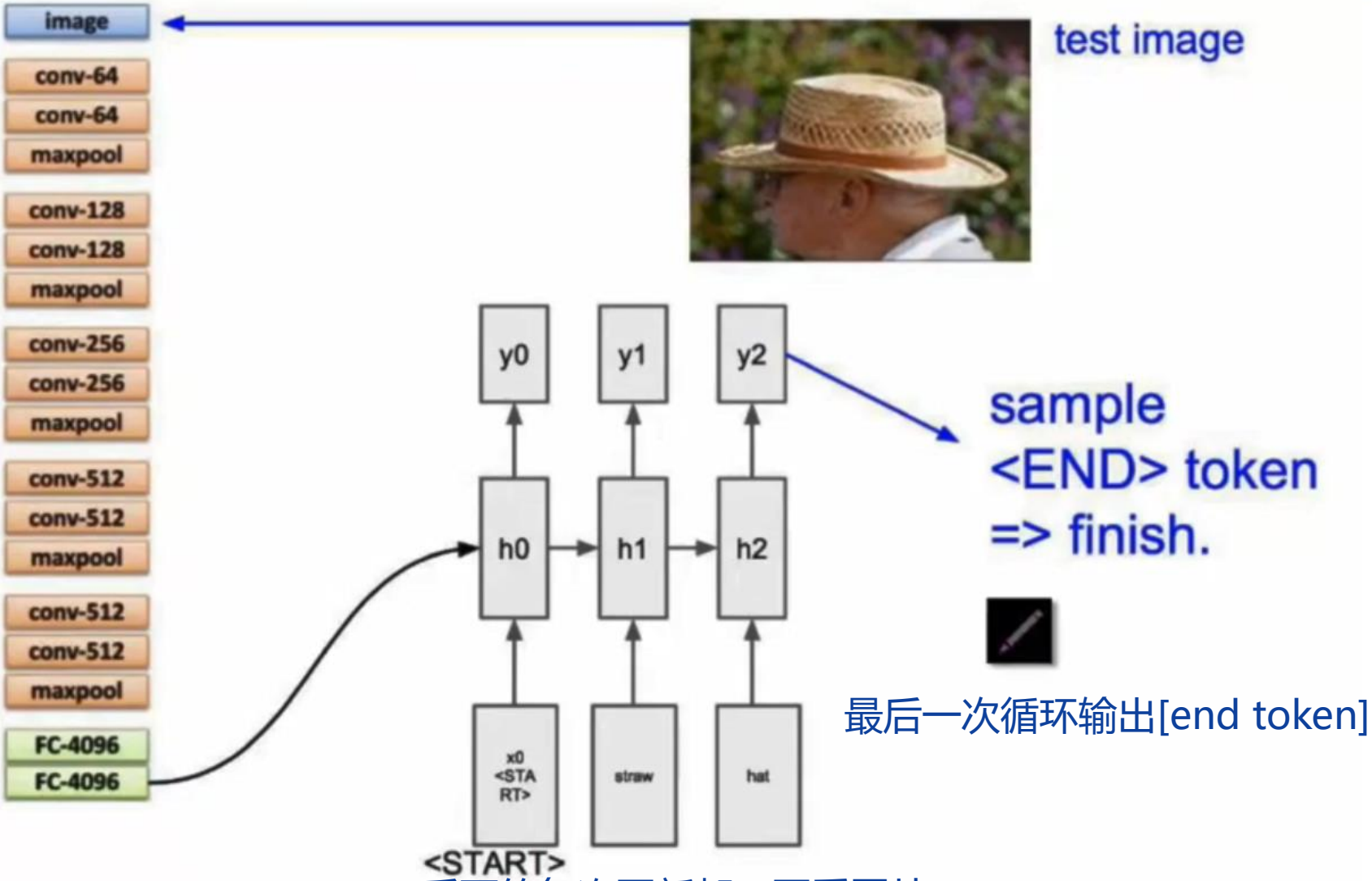
4096*1的特征向量

$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$

文本输入　记忆　图片信息

只在第一步做，后面的每次更新都不再看图片

RNN与图像描述



最后一次循环输出[end token]

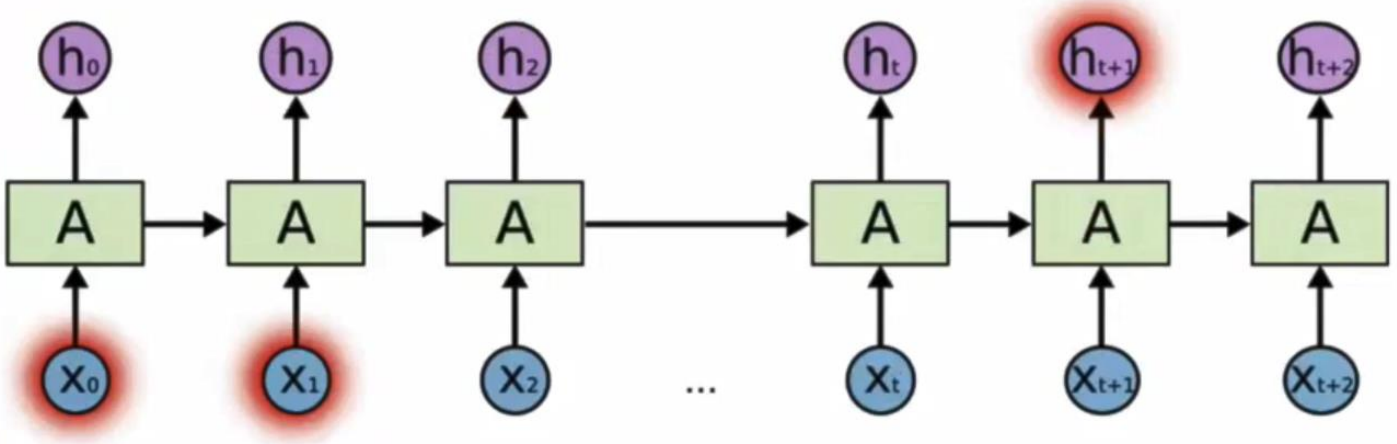
后面的每次更新都不再看图片

RNN的问题

- 相比浅层神经网络或CNN，RNN解决了之前的信息保存的问题
- 但是，RNN存在长期依赖的问题
  - 分析电影的时候，某些情节的推断需要依赖很久以前的一些细节
  - 但是，随着时间间隔不断增大，RNN会丧失学习到连接如此远的信息的能力
- 面临的问题：
  - 梯度消失问题
  - 梯度爆炸问题

RNN的问题

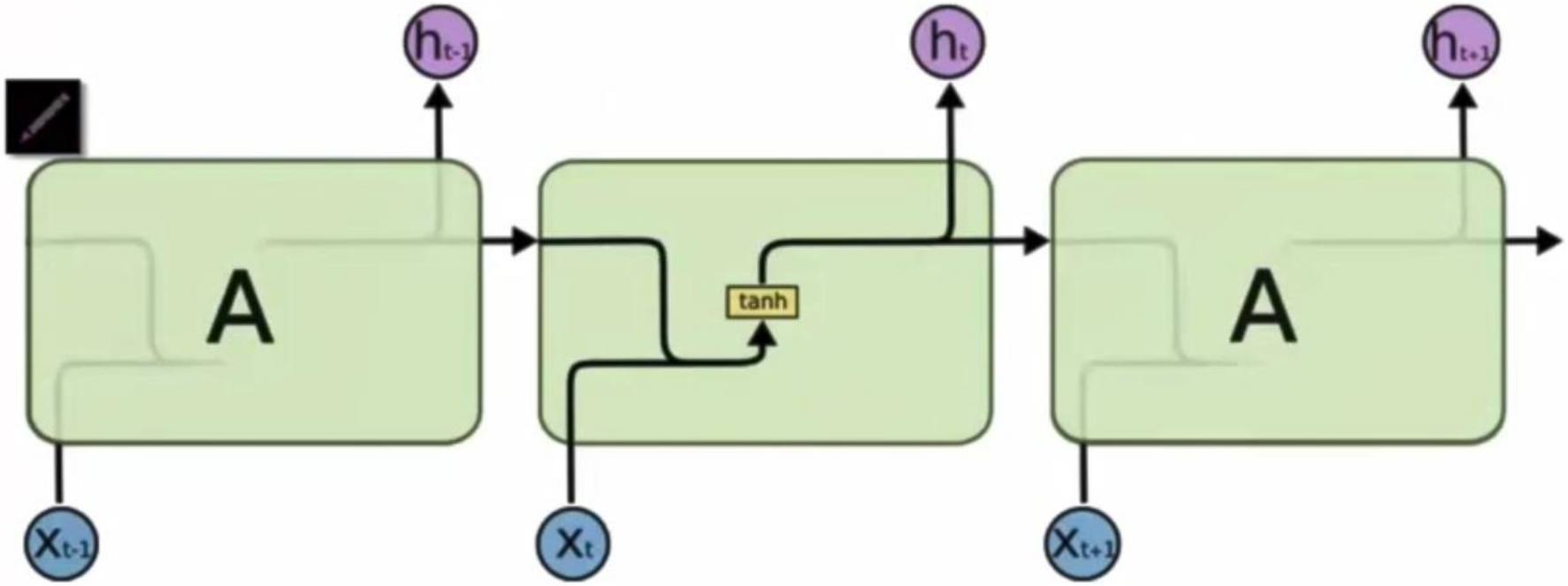
- RNN可以被训练来，通过前面的单词来预测接下来的单词。
- 实际上，相关信息和需要该信息的位置之间的距离可能非常的远。
- 不幸的是，随着距离的增大，RNN对于如何将这样的信息连接起来无能为力。



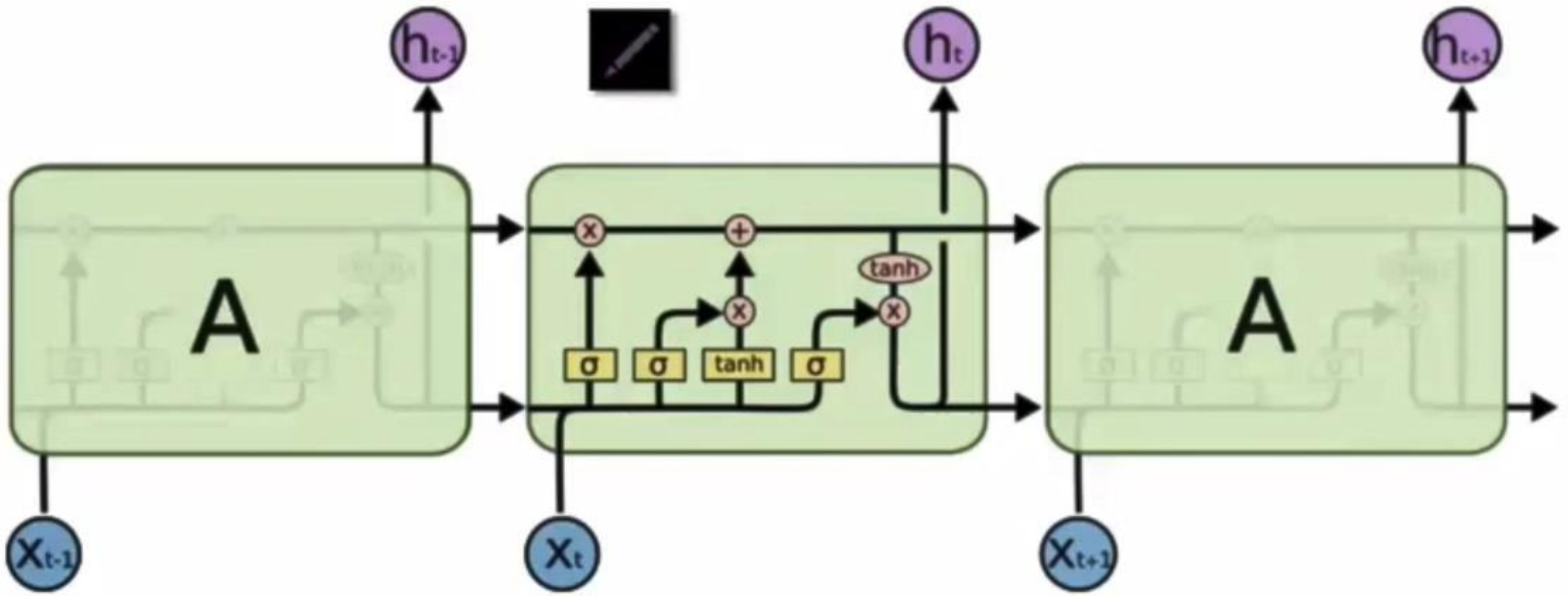
解决方案

- 选择其他的激活函数。例如ReLU。
- 引入改进网络结构的机制，例如LSTM，GRU。
  - 现在在自然语言处理上应用十分广的的就是LSTM。
- 长短期记忆网络“LSTM”是一种特殊的RNN， 它能够学习长期依赖。
  - LSTM由Hochreiter & Schmidhuber (1997) 引入，后来在很多人的努力下变得越来越精炼和流行
  - 它的“记忆细胞”被改造过
  - 该记的信息会一直传递，不该记的会被“门”截断

RNN网络



LSTM网络



Neural Network Layer
非线性处理模块

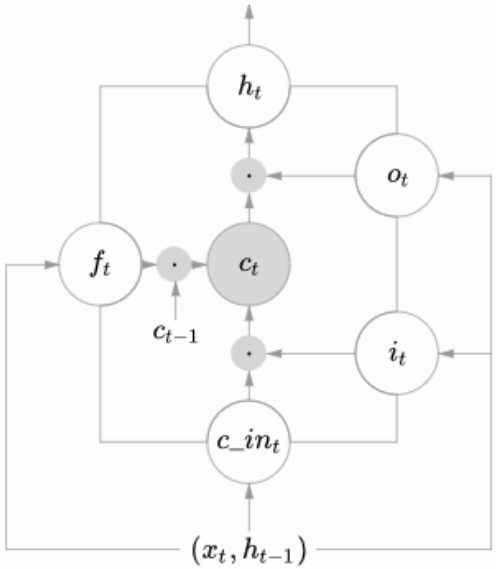
Pointwise Operation
逐点运算

Vector Transfer
信息传播方向

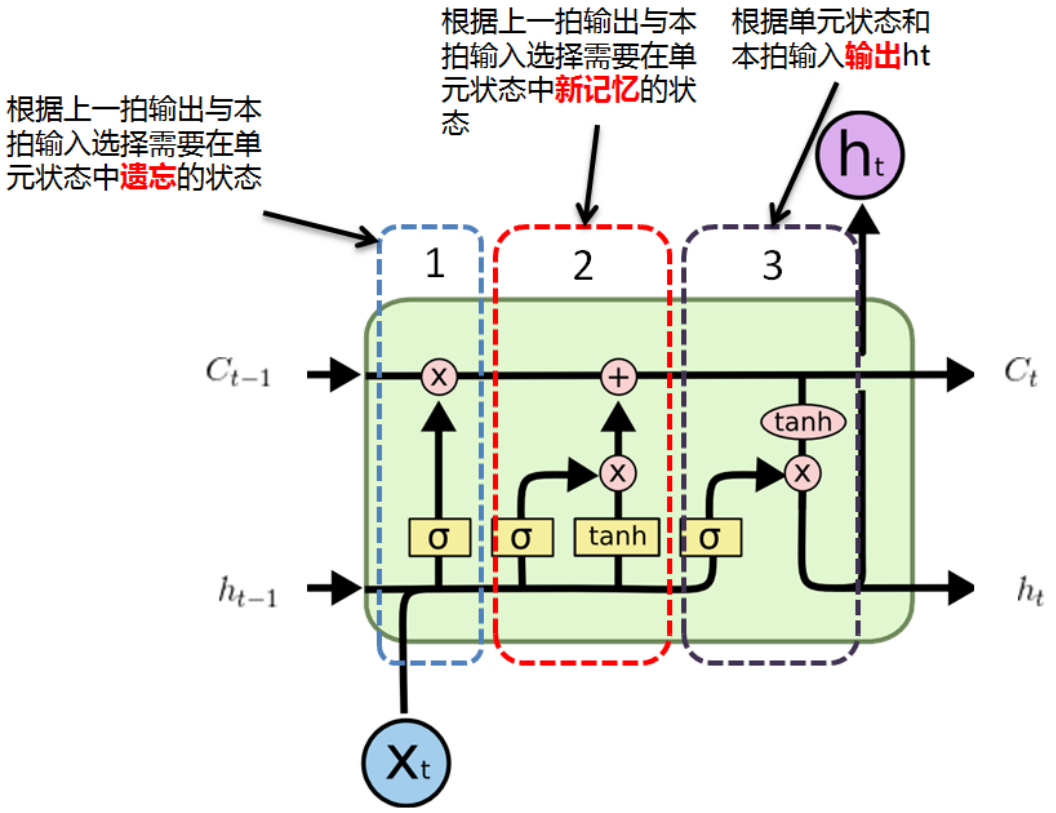
Concatenate
信息拼接

Copy
信息复制

LSTM网络



f_t：遗忘门
i_t：输入门
o_t：输出门



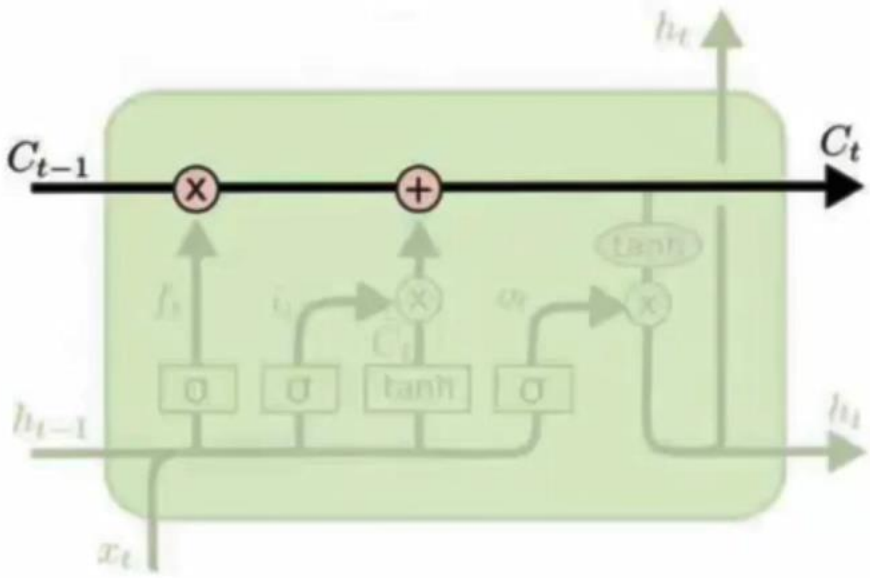
C_{t-1} C_t 单元状态 CellState

h_{t-1} h_t 单元输出

x_t 单元输入

LSTM的核心过程

- “细胞状态” Cell State
  - 细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变会很容易

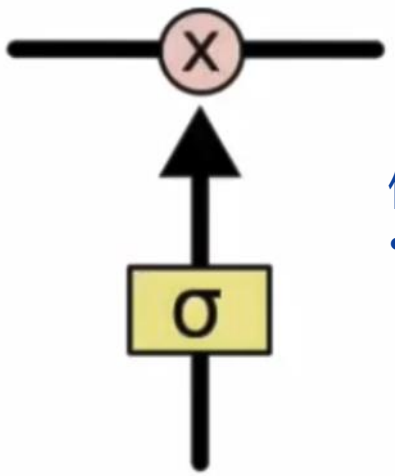


t-1时刻的记忆到现在的记忆，在传送带上往前传，发生信息的交互

- 可以取东西
- 也可以放东西上去

LSTM的核心过程

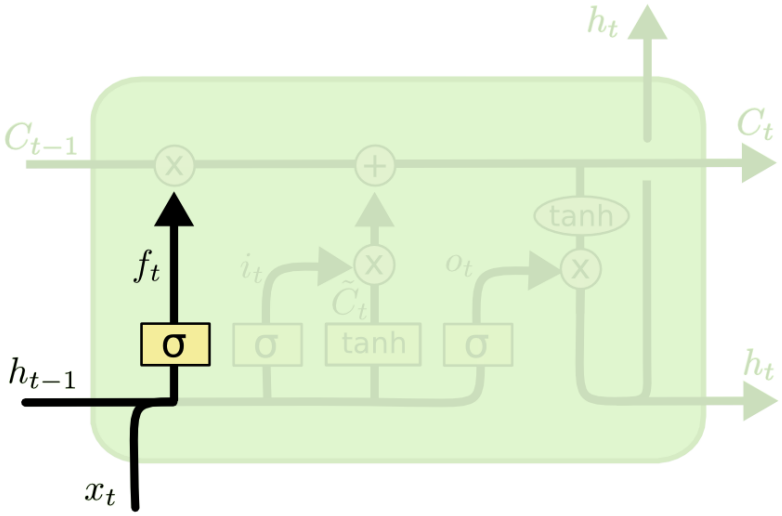
- 控制“细胞状态”
  - 通过“门”让信息选择性通过，来去除或者增加信息到细胞的状态
  - 包含一个sigmoid神经网络层，和一个逐点运算的乘法操作
  - sigmoid层输出0到1之间的概率值，描述每个部分有多少量可以通过。
    - 0代表“不允许任何量通过”
    - 1代表“允许任意量通过”



做记忆的变更
- 要把这部分记忆存下去

LSTM的核心过程

- LSTM的第一步就是决定什么信息应该被神经元遗忘。
  - 这是一个被称为“遗忘门层”的sigmoid层组成的。
  - 它输入ht−1和xt,然后在Ct−1的每个神经元状态输出0~1之间的数字。
    - “1”表示“完全保留这个”，“0”表示“完全遗忘这个”。



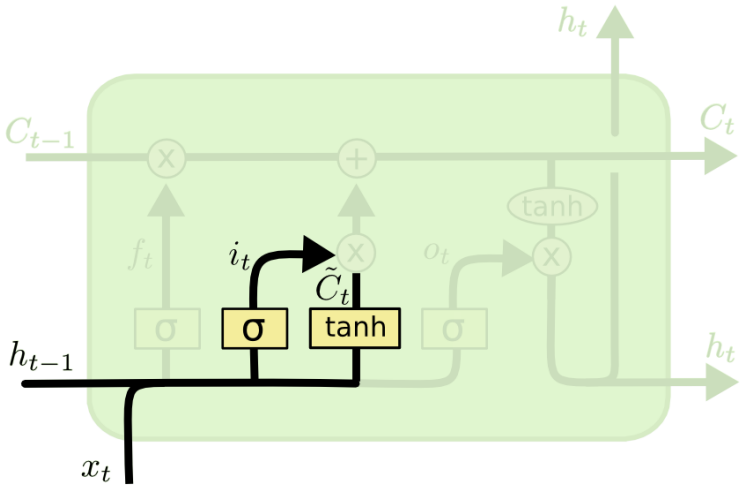
上一个时刻的输出

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

以多大概率去丢掉信息　　现在的输入

LSTM的核心过程

- 下一步就是决定我们要在神经元细胞中保存什么信息，什么新信息放到“细胞状态”中
  - 首先，一个被称为“遗忘门层”的sigmoid层决定我们要更新的数值。
  - 然后，一个tanh层生成一个新的候选数值，$\tilde{C}_t$，它会被增加到神经元状态中。
  - 这两步为状态更新做准备



产生一个概率值，以多少值去更新信息

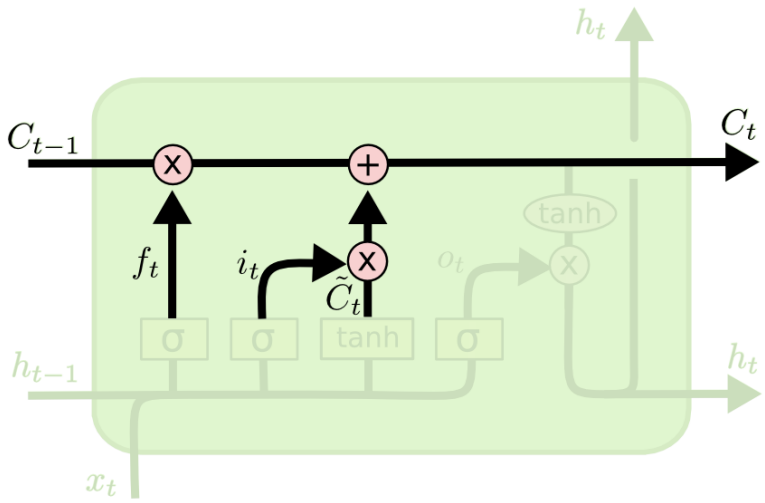
- 用i_t对C_t做过滤，哪一部分知识能够补充到我之前的知识体系中

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

LSTM的核心过程

- 更新“细胞状态”，更新C_{t-1}为C_t
  - 下一步我们给旧的状态乘以一个f_t，遗忘掉我们之前决定要遗忘的信息
  - 然后我们增加$i_t \times \tilde{C}_t$。这是新的候选值，是由我们想多大程度上更新每个状态的值来度量的。



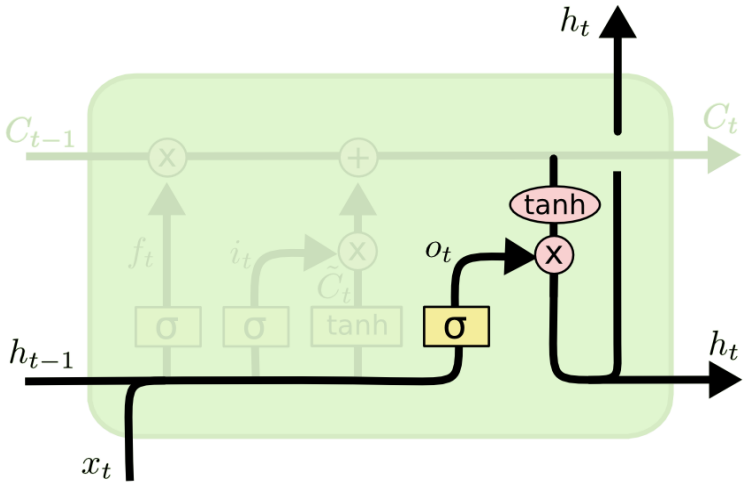
通过率 * 旧的记忆

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

筛选器 * 新的知识

LSTM的核心过程

- 基于“细胞状态” 决定输出
  - 这个输出是建立在我们的神经元状态的基础上的，但是有一个滤波器。
  - 首先，我们使用sigmoid层决定哪一部分的神经元状态需要被输出
  - 然后我们让神经元状态经过tanh（让输出值变为-1~1之间）层并且乘上sigmoid门上的输出，我们只输出我们想要输出的。



P(0,1)　　从以前的知识中筛出来
解决当前题目的知识

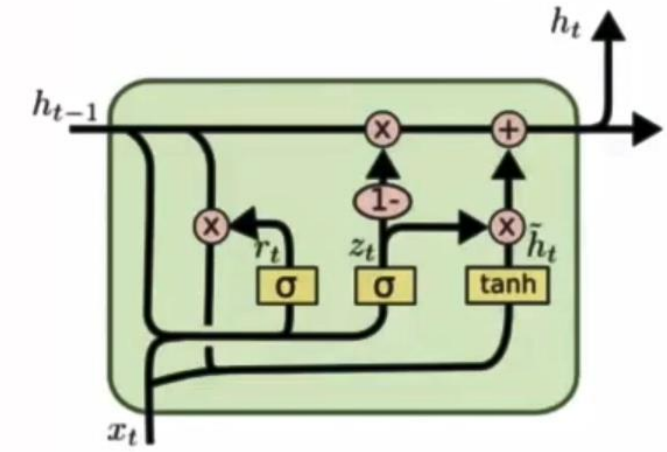
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

解决完了给出答案　　所有的知识

LSTM的变种：GRU

- Gated Recurrent Unit(GRU)，2014年提出
  - 将遗忘门和输入门合成了一个单一的“更新门”
  - 同样还混合了细胞状态和隐藏状态，以及其他的一些改动
  - z_t更新门update gate vector
  - r_t重置门reset gate vector



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

更新记忆

h_t: 忘记传递下来的h_{t-1}中的某些维度信息，并加入当前节点输入的某些维度信息

同济大学交通运输工程学院
COLLEGE OF TRANSPORTATION ENGINEERING
TONGJI UNIVERSITY

第十二讲 结束